2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

Training Data Leakage via Imperceptible Backdoor Attack

Xiangkai Yang*, Wenjian Luo*†, Qi Zhou*, and Zhijian Chen*
*Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies,
School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen 518055, Guangdong, China
†Peng Cheng Laboratory, Shenzhen 518055, Guangdong, China
Email: 21S151149@stu.hit.edu.cn, luowenjian@hit.edu.cn, 22S051036@stu.hit.edu.cn, 21B951010@stu.hit.edu.cn

***Abstract*—Recently, deep neural networks (DNNs) have been widely used and proven successful in many real-world tasks. There are many third-party DNN services available for data holders who want to develop custom DNN applications for their data and tasks. To ensure data privacy, it is crucial to safeguard the data holder's training data. This paper explores a unique attack paradigm where a hostile third-party DNN model supplier subtly obtains training data from the data holder. Prior attacks which can steal training data typically use augmented datasets to memorize the information of the data that the attacker intends to steal. However, these attacks are easily identified since the augmented datasets are visually different from the original dataset and rendered ineffective. In this attack, we generate an augmented dataset by modifying a portion of the training data using the DNN-based image steganography technique. This approach creates an augmented dataset that is visually identical to the original training dataset, making it difficult for humans to detect. Through extensive experiments, we have successfully and quietly accessed the confidential training data of data holders.**

***Index Terms*—Deep neural networks, data privacy, backdoor attack, steganography**

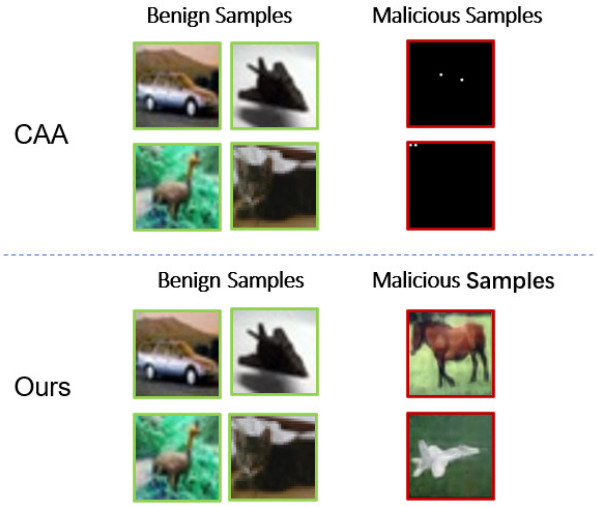

Fig. 1. The comparison of the augmented training dataset (i.e., malicious samples) in CAA and our attack.

I. Introduction

In recent years, deep neural networks (DNNs) have achieved unprecedented success in many fields, including image classification [1] and natural language processing [2], [3]. However, a significant quantity of training data, growing processing power, and complex networks determine how effective DNNs are. It is possible that non-expert data holders cannot create their networks from the ground up. As a substitute, they would use models that third parties offered. For instance, they may choose an algorithm model from an accessible algorithm marketplace that is suitable for their needs and datasets, and then train the model using the datasets they had. Even though data holders widely use this mode, it has resulted in the emergence of new privacy threats. These threats take the form of models or model-training algorithms that appear to be authentic, but purposefully leak information regarding data holders' training datasets. This is especially concerning when it comes to privacy datasets (such as a person's personal contact information, residence address, or healthcare records).

Deep neural networks (DNNs) are powerful but can pose security risks due to their high capacity. During training, DNNs can inadvertently remember specific details, potentially releasing a vast amount of knowledge about the training data.

Some examples of these types of attacks include: the membership inference attack [4], [5], which determines if a specific example was used during model training; the model inversion attack [6], [7], which recovers the values of sensitive attributes of the inputs relying on outputs from a classifier; and the property inference attack [8], [9] which draws conclusions about properties based on the fact that they are valid for a portion of the data that was used for training. In addition, the work in [10]–[12] demonstrates that malicious machine learning algorithms can produce models that fulfill the generally accepted indicators of accuracy and generalizability while at the same time leaking private information concerning their training datasets. This is true even if the adversary has limited access to the model in its black-box form. To get precise training data, they came up with the idea of the Capacity Abuse Attack (CAA), which

This study is supported by the National Key R&D Program of China (Grant No. 2022YFB3102100), Shenzhen Fundamental Research Program (Grant No. JCYJ20220818102414030), the Major Key Project of PCL (Grant No. PCL2022A03), Shenzhen Science and Technology Program (Grant No. ZDSYS20210623091809029), Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies (Grant No. 2022B1212010005). *(Corresponding author: Wenjian Luo.)*

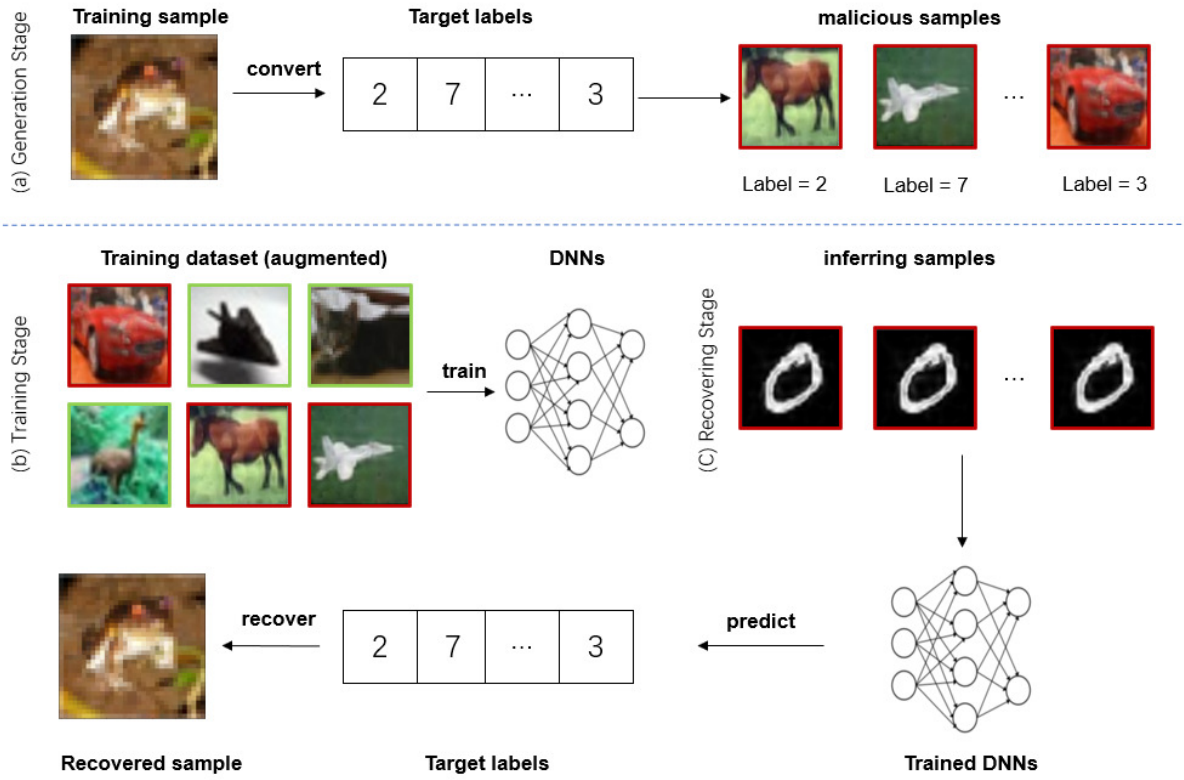

Fig. 2. The flow of our attack for stealing one training sample. In the generation stage, the attacker generates some malicious samples which are visually indistinguishable from the original training samples. However, the labels of those malicious samples encode the information of the training sample the attacker wants to steal. The union of malicious and original training samples is used to train DNNs in the training stage. In the recovering stage, the attacker generates some inferring samples by using the same way as generating malicious samples (i.e., the classes of malicious samples and inferring samples are the same, as well as the order). Since the mapping from the class of malicious sample to the target label was generated in the training stage, thus after the attacker inputs those inferring samples to the trained DNNs, the target labels, which are the encodings of a training sample, will be obtained. Thus, the attacker obtains a training sample recovered by the target labels.

adds more synthetic data to the training dataset yet does not alter the training algorithm in any way. [12] propose a novel efficient white-box backdoor attack method which leverages the linear combination of weights to remember the private data during training.

The fatal flaw of the CAA is that the synthetic augmentation data are substantially distinct from the original pictures in the training dataset. Despite the fact that CAA is the most advanced black-box attack technique currently available, CAA has this fatal flaw. Because of this, CAA is unable of evading the detection of the training data, regardless of whether the type of detection being used is an automated one or a human eye examination. Work in CAAII [13] addresses the problem which the attackers do not know the mapping relationship between the form outputted by the trained model and the label encodings in CAA, however it does not settle the problem of stealthiness in CAA. And work [14] proposed a method that the malicious data generated similar to real and natural images, improves the stealthiness of CAA, however, the backdoor triggers are still visible and recognized to human. In this regard, CAA and its existing improvements are essentially lacking in some degree of stealth. In this article, we suggest an innovative approach to the construction of the augmentation data. The following is a summary of our contributions.

- We propose a novel attack paradigm to steal training data imperceptibly from a data holder, where the constructed augmentation data have a high degree of similarity with the original training data, and the human eye cannot discern the nearly invisible difference, making our attack more covert than CAA. The comparison between the augmented training dataset in CAA and our attack is depicted in Figure 1.
- We test our method on the standard datasets MNIST, Fashion-MNIST, and CIFAR-10 to see how well it works. Experiments show that our attack could get high-quality training images with a tiny effect on the model's original task for each dataset.

II. Related Work

A. Backdoor Attack

The backdoor attack has recently emerged as a potential security risk for deep neural networks (DNNs) on image classification tasks. In this kind of attack, the attacked DNNs seem to be authentic. Still, they actively misbehave (classify to target labels that the attacker predefines) on some particular circumstances of the inputs. (i.e., poison images with attacker-specified triggers). Existing attacks can be divided

into two distinct categories according to the degree to which individuals are able to recognize their triggers. If the triggers of a backdoor attack are concealed, or if poisoned samples cannot be visually differentiated from benign ones, then the backdoor attack is invisible. If neither of these conditions is met, the backdoor attack is visible.

1) Visible Backdoor Attack: In the first backdoor attack on the task of classifying images, known as BadNets [15], the attacker introduces a backdoor by labeling benign data with the attacker's chosen target label and adding a trigger (such as a little white-black pattern at particular spots in the images). When an image with the corresponding backdoor trigger is supplied, this backdoor can be activated and output the target label during the inference step. Several further backdoor attacks have been suggested since then. Typically, [16] proposed an input-aware backdoor attack, where triggers have diversity and are nonreusable for different inputs to evade defense. [17] presented an adaptable and covert backdoor attack that avoids backdoor scanners by leveraging backdoor triggers made out of pre-existing benign properties of different labels.

2) Invisible Backdoor Attack: [18] initially emphasized the need for a backdoor attack to be invisible in order to avoid human inspection and proposed that the poisoned image should be very similar to its benign counterpart. Instead of stamping, a mixed method is employed to create poisoned images by fusing the backdoor trigger with good images. [19] recently proposed warping-based triggers which via subtle image warping achieve undetectable by machine defenders and humans. Li et al. [20] proposed an invisible backdoor attack that generates sample-specific invisible additive noises as backdoor triggers by encoding an attacker-specified string into benign images through an encoder-decoder network.

B. Capacity Abuse Attack (CAA)

The Capacity Abuse Attack (CAA) was proposed by Song et al. [10] as a method for extracting precise training data. In this attack, the attacker is a malicious machine learning (ML) model provider who is unable to access the model parameters. Still, it does have access to the prediction application programming interface (API) that was published by the data holder. Because deep learning models have such a vast capacity for memorizing, it is virtually possible for them to describe any function that is appropriate for any data. In CAA, the model is fitted not only to the original training dataset but also to the synthetic data that is labeled with the encodings of the training images which the attacker attempts to steal. This is done so that the model can accurately represent the training data. During the inference step, the attacker feeds the model the same synthetic data used during the training stage. Based on the associated outputs, the attacker can reconstruct the images utilized during the training stage.

C. Steganography

Steganography is a technique for information hiding, it does not let anyone except the intended recipient know the event of information transmission (not just the content of the information). In essence, the information-hiding process in the steganography system starts with identifying the redundant bits of cover media (those bits that can be modified without destroying the integrity of the media). StageStamp [21] is a learned steganography algorithm that can encode and decode any hyperlink bit string into a photo in a way close to perceptual invisibility. They hide data in the least significant bit of the image: subtle color and brightness changes. In [20], they train the encoder-decoder network according to the settings in StageStamp and then generate sample-specific triggers through the trained encoder. Our method uses steganography to perform a stealthy attack. In this attack, we use steganography to modify images and use them to memorize the data holder's private training data.

III. Overview

In this section, we first introduce the threat model of our proposed attack, then briefly demonstrate our attack flow.

A. Threat Model

We focus our work on image classification tasks. Our threat model is very similar to that of [10]. We assume the attacker is a dishonest third-party DNNs model provider who gives the training codes to the data holder. The data holder, who is not an expert, is going to directly adopt the training codes in order to generate tailored DNNs for the private training data, but they do not understand what the training codes are actually doing. The holder of the data will accept the modified DNNs as long as they have high predicted accuracy on the test dataset. The data holder will then expose the DNNs as an API for public or commercial services. We are going to assume that the attacker has access to the application programming interface (API) that, when fed an image, generates a probability vector that represents the likelihood that the provided input corresponds to each class.

The attack's efficacy and stealthiness are the attacker's two primary priorities. The data holder must adopt the resultant DNNs trained by the attacker's training codes for this to be successful. Additionally, the attacker needs to be able to retrieve the data holder's recognized training samples after visiting the public API and doing specific queries in order to do so. To maintain the stealthiness of the model, the data holder must detect very few exceptions in the training dataset while the model is being trained and in the requested inputs after the model has been published.

B. Attack Flow

The flow of our attack for stealing one image from the data holder is shown in Figure 2. In the generation stage, the attacker converts one specified training sample (e.g., the first sample in the training dataset) to target labels in a certain way that the attacker can flexibly design. Then the attacker generates malicious samples with a technique, which adds triggers to benign samples (i.e., some of the original training samples), and the added triggers are invisible so that

malicious samples are visually indistinguishable from benign ones. The attacker labels those malicious samples with target labels. We refer to the malicious samples generated by the same kind of trigger as the same class, so that each class of malicious samples could be used to memorize one label in target labels.

During the training stage, the data holder trained the DNNs using the conventional training process on the poisoned dataset (i.e., the union of malicious samples and the original training samples). During the recovering stage, the attacker only needs to find one arbitrary natural image (for example, one training sample in the MNIST dataset) to use to generate benign test samples. The attacker must apply the same triggers that were used during the generation stage in order to generate malicious test samples. Upon the submission of malicious test samples to the application programming interface (API) made public by the owner of the data, the attacker should be able to retrieve the labels matching the triggers and, ideally, the target labels. Lastly, the attacker recovers the training sample by using an inversed process of converting the training sample to target labels in the generation stage.

IV. Design

This section presents the proposed techniques in detail following the attack flow.

A. Generating Invisible Triggers

Inspired by StegaStamp [21] and prompted by [20], [22], we generate malicious images by adding invisible additive noises on benign images, the invisible noises embed specific strings related to the target labels using an encoder-decoder network. StegaStamp is a learned steganographic algorithm to enable robust encoding and decoding of arbitrary hyperlink bit strings into photos in a manner that approaches perceptual invisibility. The goal of StegaStamp's encoder training is to reduce the visual disparities between the original and encoded versions of a picture. The decoder is an artificial neural network that has been taught to decode messages from encoded images. The string can be designed by the attacker, and we expect the trained model can map different strings into different target labels rather than mapping different strings into one target label. In our setting, we refer a backdoor trigger as the string embedded into the image by the encoder.

B. Generating Malicious Dataset

The pseudocode for constructing a malicious dataset is described in the Algorithm 1. In line 2, **ImageToLabel**$(\cdot)$ converts the n images in the training dataset D_{train} to target labels set Y_{mal}, which is the same as the method in [10], [13], [14]. To be more specific, n is the number of images in D_{train} that the attacker intends to steal, and the attacker specifies which n training images to be encoded. Then **ImageToLabel**$(\cdot)$ represents these n images in pixels, each pixel value in the range $[0, 256)$ would be scaled to $[0, c)$ for the sake of simplicity (pixel value is not necessarily to be scaled, or it would cost several labels to encode), where c is the number of classes in D_{train}. The scaled pixels are the target labels Y_{mal}, which are the information the attacker has encoded and expected to recover after the model is published.

Algorithm 1 Generate Malicious Dataset

Input: Training dataset D_{train}, the number of malicious samples for one backdoor M, the number of stolen images n.
Output: Malicious dataset D_{mal}

1: $D_{mal} \leftarrow \emptyset$
2: $Y_{mal} \leftarrow$ **ImageToLabel**(D_{train}, n)
3: **for** each $m \in [0, M)$ **do**
4: **for** each $y_i \in Y_{mal}$ **do**
5: $x_{benign} \leftarrow$ randomly select one image in D_{train}
6: $x_i \leftarrow$ **Encode**$(x_{benign}, string(i))$
7: $D_{mal} \leftarrow D_{mal} \cup (x_i, y_i)$
8: **end for**
9: **end for**
10: **Return** D_{mal}

Lines 4-8 of the code generate one malicious image for each target label in Y_{mal}. Specifically, **Encode**$(\cdot)$ accepts one benign image and one string as parameters and then returns a malicious image that hides string information in it while having few discernible variations from the benign image. It is noted that **Encode**$(\cdot)$ is implemented by a modified version of StegaStamp [21]. We set up the index i of y_i in Y_{mal} as a string, so each target label in Y_{mal} corresponds to one unique string. Thus, the mapping from the one-of-a-kind string to the target label is generated and could be learned during training.

Aside from that, similar to the backdoor attack, learning the mapping from trigger to target label typically requires several poisoned samples. We set up M as the number of malicious samples for each target label. Malicious samples of different target label corresponds to a different string. However, it does not matter which benign image is chosen to generate the malicious image, so we randomly select one image in D_{train} as a benign image for generating each malicious image.

Lines 3-9 of the code generate M malicious images for each target label in Y_{mal}. In general, the greater the value of M is, the more likely the model learns the relationship between one string and the target label, which encodes the training set information. Meanwhile, the increase of M will also decrease the attack's stealthiness since the poisoning rate γ (i.e., the ratio of the size of malicious samples and training samples) increases, and it might have an unmissable impact on the benign test accuracy. So the attacker usually has a trade-off between the attack effectiveness and stealthiness when setting up M.

C. Recovering Images

Algorithm 2 describes the pseudocode for recovering images. **NumOfInput**$(\cdot)$ accepts the quantity of stolen images n and the attacker's auxiliary knowledge of the training data Aux (the size of the training images) as parameters, then

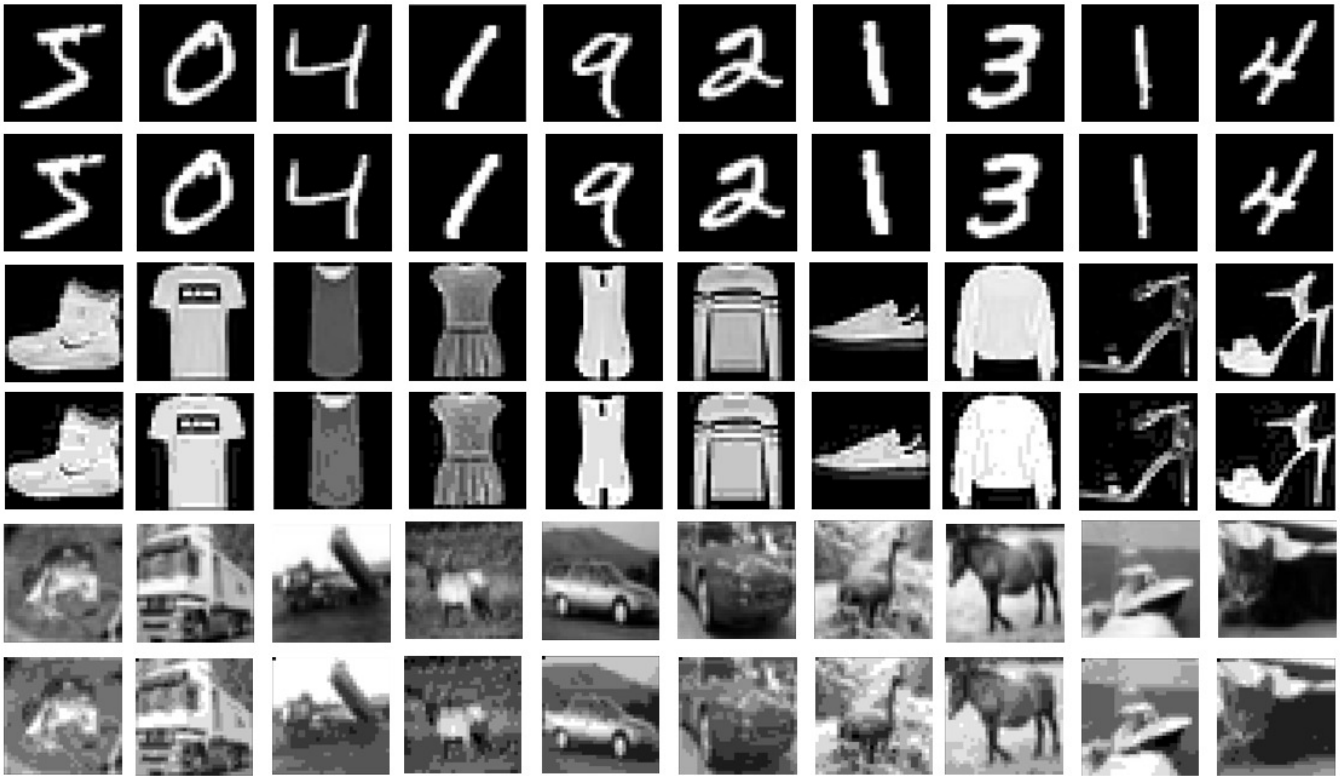

Fig. 3. The comparison of the original images and the recovered images of ten times experiments (stealing different images each time) on three datasets. The first row contains the original images, while the second row contains the MNIST experiment's recovered images. The original and recovered images in the Fshion-MNIST experiment are located in the third and fourth rows, respectively. In the CIFAR-10 experiment, the original images are located in the fifth row, while the recovered images are located in the sixth row.

Algorithm 2 Recover Images

Input: The auxiliary knowledge of training data Aux, trained model θ, the number of stolen images n.
Output: Recovered images $recoveredImg$

1: $n_{test} \leftarrow$ **NumOfInput**(n, Aux)
2: $x \leftarrow$select a random image
3: $Y_{decode} \leftarrow \emptyset$
4: **for** each $i \in [0, n_{test})$ **do**
5: $\quad x_i \leftarrow$ **Encode**$(x, string(i))$
6: $\quad y_i \leftarrow \theta(x_i)$
7: $\quad Y_{decode} \leftarrow Y_{decode} \cup (i, y_i)$
8: **end for**
9: $recoveredImg \leftarrow$**RecoverImage**(Y_{decode}, Aux)
10: **Return** $recoveredImg$

return the quantity of input test images n_{test}, and n_{test} should equal size of Y_{mal} in Algorithm 1. The code in lines 4 through 8 generates n_{test} malicious test images, each of which is utilized to infer a target label based on its encoded string. Given that the mapping from the string to the target label is created during training, the attacker should ideally be able to determine the target label associated with the encoded string when they submit a malicious test image to the trained model θ. The last step, **RecoverImage(·)** converts Y_{decode} into images according to Aux, which is an inverse process of **ImageToLabel**(·) in Algorithm 1. These images are training images that the attacker intends to steal from the data holder.

V. Experiments

In this section, first, we introduce our experimental setting, including datasets, models, the attack setup, and evaluation metrics. Then we demonstrate experimental results. The source codes are available at https://github.com/MiLab-HITSZ/2023YangIBA.

A. Experimental Setting

1) Datasets and Models: We conducted experiments on three classical image classification datasets MNIST [23], Fashion-MNIST [24], and CIFAR10 [25]. The MNIST dataset is comprised of 60,000 training samples and 10,000 test examples. Each sample is a 28-pixel square grayscale image categorized under one of ten categories, with 6,000 examples per category. Fashion-MNIST, on the other hand, presents a more formidable challenge for categorization. It includes 60,000 training samples and 10,000 test samples, with each instance represented by a grayscale image of 28 by 28 and assigned a label from ten categories. CIFAR-10 is another dataset used for this purpose, with ten different types of RGB graphics, each measuring 32 by 32 by 3. There are 6000 images in each class, with 5000 used for training and 1000 for testing.

We utilize the LeNet-5 [23] model to construct the classifier that will be used on the MNIST and Fashion-MNIST datasets. For CIFAR-10 image categorization, we use the ResNet-18 [1] model. Every single training classifier is run through the SGD optimizer, and all of the optimizer's hyperparameter settings are the same. The weight loss rate is 0.0001, the momentum is 0.9, and the learning rate is 0.01. The maximum epoch and the batch size have been set to 100 and 256, respectively.

2) Attack Setup: We train an encoder-decoder on the ImageNet [26] dataset and use the encoder as an invisible malicious samples generator for all of our experiments. We modify the setting in StegaStamp [21] for our purpose. Specifically, we get rid of operations that apply image perturbations between the encoder and decoder for real-world robustness, and those operations including perspective warp, motion and defocus blur, color manipulation, noise, and JPEG compression. Those operations are not necessary in our attack circumstance. We set the number of target images n as 1. We resize the size of dataset images to 224 × 224 × 3, and we modify the corresponding models as well. The reasons is that, for the sake of simplicity, we use pre-trained models (i.e., LeNet-5 for MNIST and Fashion-MNIST, ResNet-18 for CIFAR-10) which provided by Pytorch as classifiers, which are commonly trained with image size 224 × 224 × 3, and we can easily obtain high precision when training models with size 224 × 224 × 3. However, the size of the target image we intend to steal is still the original size rather than resized size (i.e., 28 × 28 for MNIST and Fashion-MNIST, 32 × 32 × 3 for CIFAR10). Besides, for simplicity, we convert the target image of CIFAR-10 to a gray-scale image. Each pixel in the target image is scaled and ranges from 0 to 9, and is encoded with one label. The quantity of malicious samples for one pixel M is set as 20. So for the MNIST and the Fashion-MNIST, the size of the malicious dataset is 15680, and the poisoning rate $\gamma = 26.13\%$. For CIFAR-10, the size of the malicious dataset is 20480, and the poisoning rate $\gamma = 40.96\%$.

3) Evaluation Metrics: We use the *mean absolute pixel error* (MAPE) [10] to measure the quality of the image we recovered. MAPE is $\frac{1}{k}\sum_{i=1}^{k}\left|x_i - x_i^{'}\right|$ when given a stolen image $x_i^{'}$ and the original image x_i with k pixels. It ranges from 0 to 255. The smaller the value of MPAE, the more effective the method is. In addition, the impact of introducing malicious datasets on benign test accuracy is considered to evaluate the attack's effectiveness and stealthiness.

B. Experimental Results

1) Attack Effectiveness: The images recovered from all three datasets, as depicted in Figure 3, are of high quality and easily recognizable. Our experiment, as shown in Figure 4, demonstrates that our attack is capable of successfully stealing an image while maintaining a low MAPE. This is achieved by malicious poisoning only a portion of the training data (26.13% for MNIST and Fashion-MNIST, 40.96% for CIFAR-10). Our results indicate that the MAPE consistently converged to 2.6, 7.4, and 14.3 during the MNIST, Fashion-MNIST, and CIFAR-10 experiments, respectively. This shows the high stability and robustness of our attack.

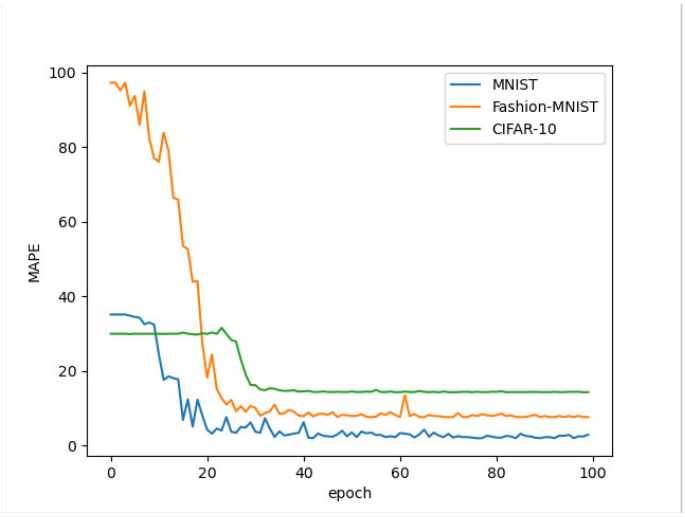

Fig. 4. The MAPEs of MNIST, Fashion-MNIST, and CIFAR-10.

2) Attack Stealthiness: As shown in Figure 5, it is evident that despite being subjected to an attack, the accuracy of the benign test only marginally decreases upon the introduction of a malicious dataset. The Fashion-MNIST dataset shows a reduction of less than 1%, while MNIST and CIFAR-10 exhibit a decline of only 0.5%.

3) Statistical Analysis: Statistical analysis of MAPE and benign test accuracy on MNIST, Fashion-MNIST, and CIFAR-10 after conducting 30 independent experiments for each dataset is shown in Table I. Specifically, first, we specify which image in the training data is to be stolen, and we conduct 3 independent experiments without changing any setup. Then, we specify another image in the training data to steal and repeat this procedure ten times until we experiment 30 times. Then we compute the average and standard deviation of MAPE and benign test accuracy for each dataset. The results show that our attack is steady on all datasets we experiment with and have few difference in MAPE compared with CAA. Although the benign test accuracy is slightly lower than CAA since we added more malicious samples (20 times than CAA), we achieved roughly the same attack effectiveness as CAA.

TABLE I
STATISTICAL ANALYSIS OF MAPE AND BENIGN TEST ACCURACY

Method	Dataset	MAPE	Benign test accuracy (%)
Ours	MNIST	2.55 ± 0.58	98.71 ± 0.16
	Fashion-MNIST	7.62 ± 2.25	91.51 ± 0.39
	CIFAR-10	14.29 ± 1.04	94.55 ± 0.23
CAA	MNIST	1.78 ± 0.56	98.78 ± 0.13
	Fashion-MNIST	7.20 ± 2.16	91.68 ± 0.32
	CIFAR-10	13.11 ± 1.02	94.93 ± 0.38

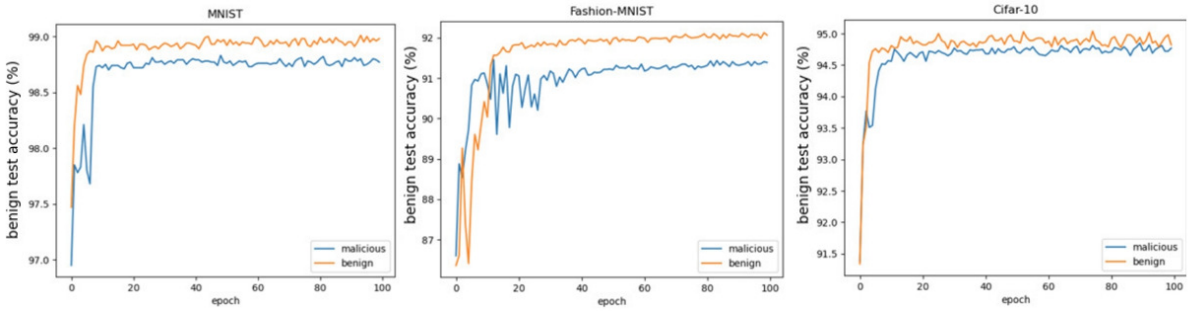

Fig. 5. The benign test accuracy (%) comparison before and after the introduction of malicious datasets.

VI. Conclusion

Using augmentation data that are visually identical to the original training data, we investigated a unique technique in this research to steal training data surreptitiously from data holders. In particular, we generate augmentation data by embedding strings into training images using a DNN-based steganography encoder; the mapping from attacker-specified strings to target labels can be produced during training. Thus, during the inference stage, we can generate malicious test images by adding known strings to an arbitrary image. Then, after submitting the malicious images to the API published by the data holders', we can obtain target labels, and the privacy training data of the data holder can be recovered in accordance with the target labels. Many tests are run, and the results show that our method can successfully and covertly get private training data from data owners.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[3] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee, 2013, pp. 6645–6649.

[4] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 3–18.

[5] Y. Long, V. Bindschaedler, L. Wang, D. Bu, X. Wang, H. Tang, C. A. Gunter, and K. Chen, "Understanding membership inferences on well-generalized learning models," *arXiv preprint arXiv:1802.04889*, 2018.

[6] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1322–1333.

[7] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton, "A methodology for formalizing model-inversion attacks," in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. IEEE, 2016, pp. 355–370.

[8] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici, "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *International Journal of Security and Networks*, vol. 10, no. 3, pp. 137–150, 2015.

[9] M. P. Parisot, B. Pejo, and D. Spagnuelo, "Property inference attacks on convolutional neural networks: Influence and implications of target model's complexity," *arXiv preprint arXiv:2104.13061*, 2021.

[10] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security*, 2017, pp. 587–601.

[11] N. Xu, Q. Liu, T. Liu, Z. Liu, X. Guo, and W. Wen, "Stealing your data from compressed machine learning models," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[12] W. Luo, L. Zhang, P. Han, C. Liu, and R. Zhuang, "Taking away both model and data: Remember training data by parameter combinations," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 6, pp. 1427–1437, 2022.

[13] W. Luo, L. Zhang, Y. Wu, C. Liu, P. Han, and R. Zhuang, "Capacity abuse attack of deep learning models without need of label encodings," *IEEE Transactions on Artificial Intelligence*, pp. 1–13, 2023.

[14] X. Yang, W. Luo, L. Zhang, Z. Chen, and J. Wang, "Data leakage attack via backdoor misclassification triggers of deep learning models," in *2022 4th International Conference on Data Intelligence and Security (ICDIS)*, 2022, pp. 61–66.

[15] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[16] T. A. Nguyen and A. Tran, "Input-aware dynamic backdoor attack," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3454–3464, 2020.

[17] Y. Li, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[18] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[19] A. Nguyen and A. Tran, "Wanet–imperceptible warping-based backdoor attack," *arXiv preprint arXiv:2102.10369*, 2021.

[20] Y. Li, Y. Li, B. Wu, L. Li, R. He, and S. Lyu, "Invisible backdoor attack with sample-specific triggers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 16 463–16 472.

[21] M. Tancik, B. Mildenhall, and R. Ng, "Stegastamp: Invisible hyperlinks in physical photographs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2117–2126.

[22] S. Li, M. Xue, B. Z. H. Zhao, H. Zhu, and X. Zhang, "Invisible backdoor attacks on deep neural networks via steganography and regularization," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2088–2105, 2020.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[25] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.